ProtoObject
 Object
  CTKeyedValuedOrderedCollection
   CTNotebookKeyedValuedOrderedCollection
    CTClassCommentAutoNotebook
     BookletDSstPdfNotebook

1. PDF files in Pharo

In order to render PDF files in a Pharo image we defined a new class `RSPdf`;
in what follow we describe how that class can be used and some important properties.

First, create a new Roassal's shape to reference a PDF file:

```
(pdf := RSPdf new
            fileReference:
               '/home/mn/Downloads/organizing-programs.pdf' asFileReference;
            pageNumber: 10;
            yourself)
```

Now the `pdf` name refers to a Roassal shape denoting the 10th page of the desired PDF; ignore how it is defined internally.

second, let us show that 10th page

```
RSGroup new
   add: pdf;
   yourself
```

ProtoObject
 Object
  Collection
   SequenceableCollection
    OrderedCollection
     RSGroup

1

46 UNGAR, CHAMBERS, CHANG, AND HÖLZLE

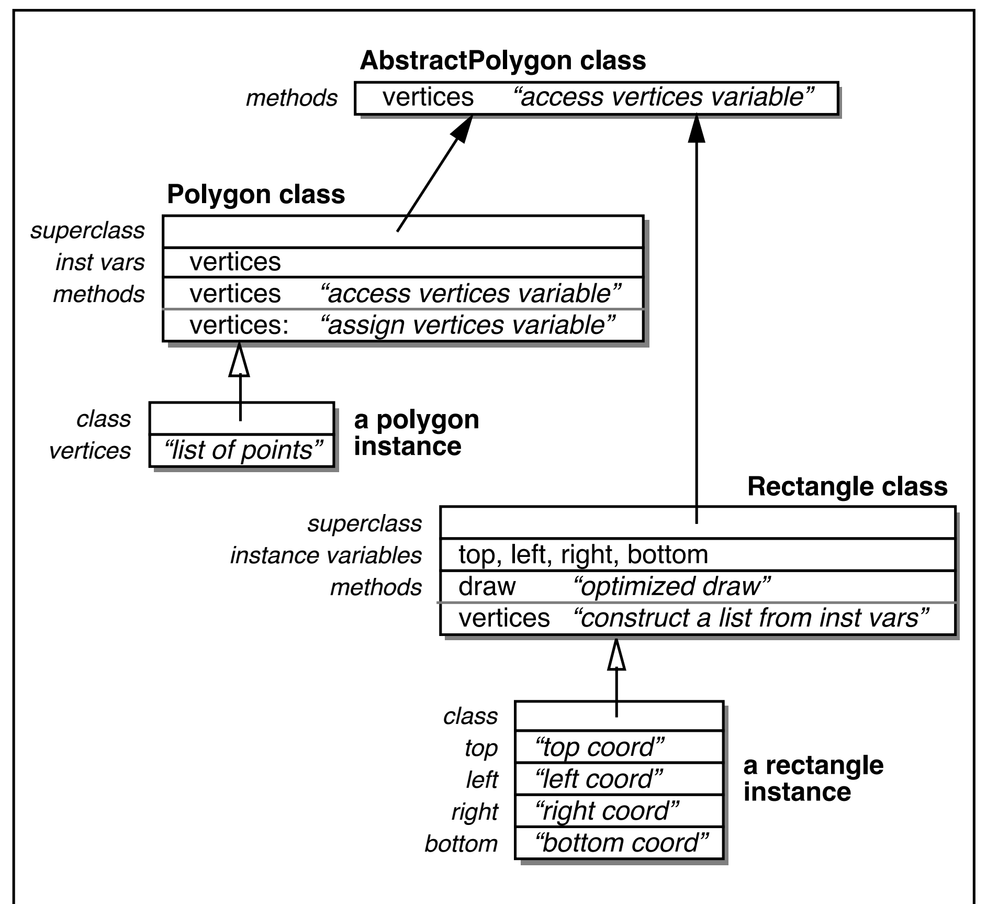

Figure 4a. Representation modification in a class-based language.

sentations (as in the filled polygon example) or not (as in the rectangle example). Both are natural and structured programming styles fostered by classless languages. Class-based languages typically have a much more difficult time handling cases that differ from strict representation extension. As mentioned above, Trellis/Owl is one notable exception. Languages with powerful metaclass facilities, such as CLOS [1], are able to define metaclasses for subclasses that do not inherit the instance variables of their superclasses, but this solution is much more complex and probably more verbose than the simple solution in classless languages.

as required.

1.1. Encompassing rectangles

The encompassing rectangle of `pdf` can be found by

```
pdf encompassingRectangle extent
```

ProtoObject
 Object
  Point

x
Answer the x coordinate. 612.0
(100@200) x >>> 100
y
Answer the y coordinate. 792.0
(100@200) y >>> 200

easily. That message rely on the low-level message

RSPdf, protocol *accessing*.

```
RSPdf >> #computeEncompassingRectangle

computeEncompassingRectangle

   | rect |
   self withPopplerPageHandlerDo: [ :popplerPageHandler |
      | w h liblua |
      liblua := LibLua uniqueInstance.
      liblua withOpenedLibsStateDo: [ :L |
         liblua on: L assertLUAOK: [
            liblua
               lual_requiref: L name: 'cairo';
               on: L push: #cairo;
               lua_getfield: L at: -1 name: 'poppler_page_get_size';
               on: L push: popplerPageHandler;
               lua_pcall: L nargs: 1 nresults: 2 ].

         w := liblua on: L at: -2.
         h := liblua on: L at: -1.

         rect := Rectangle center: 0 @ 0 extent: w @ h ] ].

   ^ rect
```

in turn. We can double check the return value of the latter message

```
extent := pdf computeEncompassingRectangle extent
```

ProtoObject
 Object
  Point

x
Answer the x coordinate. 612.0
(100@200) x >>> 100
y
Answer the y coordinate. 792.0
(100@200) y >>> 200

which divided by 72 (*perhaps the DPI used by the underlying library*) yields the size in centimeters,

```
extent / 72 * 2.54
```

ProtoObject
 Object
  Point

x
Answer the x coordinate. 21.59
(100@200) x >>> 100
y
Answer the y coordinate. 27.94
(100@200) y >>> 200

to be compare with the usual A4's size of 21 times 29.7 centimeters.

1.2. Under the hood

According to the *official documentation* [1] we show our calling message,

[1] https://poppler.freedesktop.org/api/glib/poppler-Poppler-Page.html#poppler-page-render

RSPdf, protocol *as yet unclassified*.

```
RSPdf >> #poppler_page_render:cairo:

poppler_page_render: p cairo: cr

   ^ self
      ffiCall:
      #( void poppler_page_render #( void #* p #, void #* cr ) )
      module: PopplerLibrary
```

where the argument `p` is a *poppler document* and the argument `cr` is a *cairo canvas* (of C type `cairo_t`).